|  |  |
| --- | --- |
| Gerb-BMSTU_01 | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное бюджетное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

ФАКУЛЬТЕТ **Информатика и системы управления**

КАФЕДРА **ПРОГРАМНОЕ ОБЕСПЕЧЕНИЕ ЭВМ И ИНФОРМАЦИОННЫЕ ТЕХНОЛОГИИ (ИУ7)**

НАПРАВЛЕНИЕ ПОДГОТОВКИ **09.04.03 ПРОГРАММНАЯ ИНЖЕНЕРИЯ**

**Отчет**

|  |  |
| --- | --- |
| **По лабораторной работе №** | 3 |

**Название:**

Алгоритмы сортировки

**Дисциплина:** Анализ Алгоритмов

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Студент | ИУ7-52Б |  |  | Н.А. Гарасев |
|  | (Группа) |  | (Подпись, дата) | (И.О. Фамилия) |
|  |  |  |  |  |
| Преподаватель |  |  |  | Л.Л. Волкова |
|  |  |  | (Подпись, дата) | (И.О. Фамилия) |

Москва, 2020

Оглавление

[Введение 3](#_Toc52317386)

[1. Аналитическая часть 4](#_Toc52317389)

[2. Конструкторская часть 6](#_Toc52317390)

[2.1. Схемы алгоритмов 6](#_Toc52317391)

[3. Технологическая часть 10](#_Toc52317392)

[3.1. Реализация алгоритмов 10](#_Toc52317393)

[3.2. Тестирование функций 12](#_Toc52317394)

[4. Экспериментальная часть 13](#_Toc52317395)

[4.1. Интерфейс 13](#_Toc52317396)

[4.2. Сравнение алгоритмов по времени работы реализаций 15](#_Toc52317397)

[Заключение 21](#_Toc52317398)

[Список литературы 22](#_Toc52317399)

# Введение

Цель лабораторной работы: изучить и применить на практике алгоритмы сортировки массивов, а также проанализировать трудоемкость этих алгоритмов и затрачиваемые ресурсы. В данной лабораторной работе рассматривается сортировка пузырьком, сортировка вставкой и сортировка выбором.

Алгоритм сортировки — это алгоритм для упорядочивания элементов в списке. Сортировка элементов достаточно распространенная проблема, которая встречается практически в каждой отрасли современного мира [1].

## В ходе выполнения лабораторной работы требуется решить следующие задачи.

* 1. Изучить выбранные алгоритмы сортировки.
  2. Реализовать алгоритм сортировки пузырьком.
  3. Реализовать алгоритм сортировки выбором.
  4. Реализовать алгоритм сортировки вставкой.
  5. Дать теоретическую оценку всем вышеупомянутым алгоритмам.
  6. Сравнить алгоритмы по затраченным ресурсам.
  7. Проанализировать полученные результаты и сделать вывод о эффективности реализованных алгоритмов.

# Аналитическая часть

Сортировка массивов — одна из основных операций над массивами [1].

## Сортировка пузырьком

Алгоритм состоит из повторяющихся проходов по сортируемому массиву. За каждый проход элементы последовательно сравниваются попарно и, если порядок в паре неверный, выполняется обмен элементов. Проходы по массиву повторяются **N-1** раз или до тех пор, пока на очередном проходе не окажется, что обмены больше не нужны, что означает — массив отсортирован. При каждом проходе алгоритма по внутреннему циклу, очередной наибольший элемент массива ставится на своё место в конце массива рядом с предыдущим «наибольшим элементом», а наименьший элемент перемещается на одну позицию к началу массива («всплывает» до нужной позиции, как пузырёк в воде — отсюда и название алгоритма).

## Сортировка выбором

Шаги алгоритма:

* находим номер минимального значения в текущем списке;
* производим обмен этого значения со значением первой неотсортированной позиции (обмен не нужен, если минимальный элемент уже находится на данной позиции);
* теперь сортируем хвост списка, исключив из рассмотрения уже отсортированные элементы.

## Сортировка вставкой

В начальный момент отсортированная последовательность пуста. На каждом шаге алгоритма выбирается один из элементов входных данных и помещается на нужную позицию в уже отсортированной последовательности до тех пор, пока набор входных данных не будет исчерпан. В любой момент времени в отсортированной последовательности элементы удовлетворяют требованиям к выходным данным алгоритма.

# Конструкторская часть

На вход алгоритмы принимают массив и его длину. На выходе выдают отсортированный по возрастанию массив.

# Схемы алгоритмов

На рис. 2-4 приведены схемы алгоритмов:

1. сортировка пузырьком;
2. сортировка выбором;
3. сортировка вставкой.

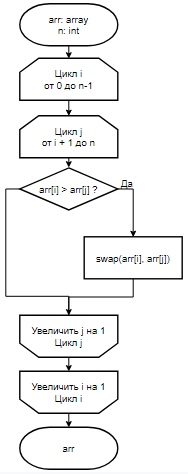


Рисунок 1. *Сортировка пузырьком*

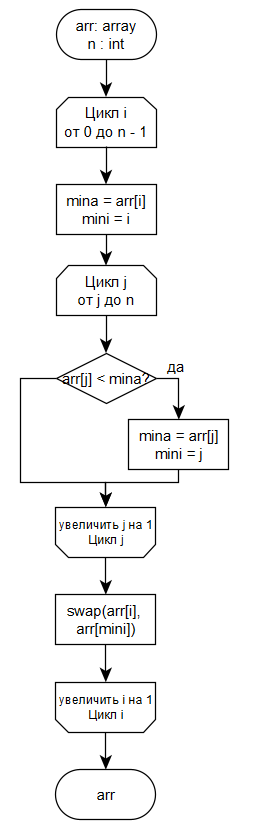


Рисунок 2. *Сортировка выбором*

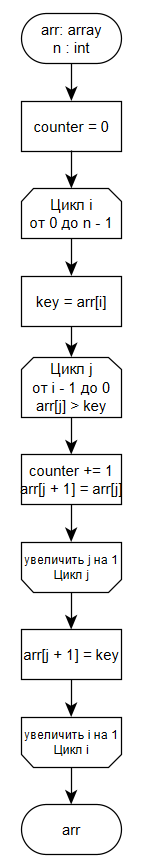
**

Рисунок 3. *Сортировка вставками*

* 1. **Оценка трудоемкости алгоритмов**

Трудоемкость алгоритма — это количество операций, которые необходимо выполнить в процессе решения поставленной задачи с помощью этого алгоритма. Рассмотрим две трудоемкости для каждого из алгоритмов: наихудшая и наилучшая трудоемкость. Наилучшим вариантом считается – вариант, где на вход поступает уже отсортированный массив. Наихудший – на вход поступает отсортированный в обратную сторону массив.

Введем модель вычисления трудоемкости для оценки алгоритмов:

* базовые операции стоимостью 1 — +, -, \*, /, =, ==, <=, >=, !=, +=, [], получение полей класса;
* оценка трудоемкости цикла: Fц = init + N\*(a + Fтела + post) + a, где a - условие цикла, init - предусловие цикла, post - постусловие цикла;
* стоимость условного перехода применим за 0, стоимость вычисления условия остаётся.
  + 1. **Сортировка пузырьком**

Рассмотрим трудоемкость стандартного алгоритма сортировки – сортировка пузырьком. В наихудшем варианте добавляется перестановка элементов

Наилучшая трудоемкость: 2 + (n - 1) \* (3 + 1 + 2 + (n / 2) \* (2 + 3)).

Итог: 2.5 \* n \* n + 2.5 \* n – 3.

Наихудшая трудоемкость: 2 + (n - 1) \* (2 + 1 + 2 + (n / 2) \* (2 + 3 + 7)).

Итог: 6 \* n \* n - 1 \* n – 3.

* + 1. **Сортировка выбором**

Рассмотрим трудоемкость алгоритма сортировка выбором. В наихудшем варианте добавляется установка нового минимума.

Наилучшая трудоемкость: 2 + (n - 1) \* (3 + 11 + 3 + (n / 2) \* (2 + 2)).

Итог: 2 \* n \* n + 15 \* n – 15.

Наихудшая трудоемкость: 2 + (n - 1) \* (3 + 11 + 3 + (n / 2) \* (2 + 2 + 3)).

Итог: 3.5 \* n \* n + 13.5 \* n – 15.

* + 1. **Сортировка вставкой**

Рассмотрим трудоемкость алгоритма сортировка вставкой.

Наилучшая трудоемкость: 2 + n \* (2 + 2 + 3 + 6)

Итог: 13 \* n + 2

Наихудшая трудоемкость: 2 + n \* (2 + 2 + 3 + 6 + (n - 1) / 4 \* (5 + 4))

Итог: 4.5 \* n \* n + 8.5 \* n + 2

Произвольная трудоемкость: 2 + n \* (2 + 2 + 3 + 6 + (n – 1) / 2 \* (5 + 4))

Итог: 2.25 \* n \* n + 10.75 \* n + 2

**Вывод:** трудоемкость напрямую зависит от входных данных. Заметим, что трудоемкость прямо пропорциональна времени выполнения алгоритмов. С помощью проанализированных трудоемкостей можно грамотно выбрать необходимый алгоритм для поставленной задачи.

# Технологическая часть

В качестве языка программирования был выбран python, т.к. данный язык программирования имеет большое количество полезных библиотек для различных необходимостей, а также язык предоставляет средства для быстрого прототипирования и динамической семантики. Для замера процессорного времени была использована функция process\_time(), стандартной библиотеки python – time [2].

## Реализация алгоритмов

В листингах 1-3 представлена реализация алгоритмов сортировки массивов. В листинге 4 представлена функция для замера времени выполнения заданной функции на заданном количестве итераций на матрицах указанного размера.

Листинг 1. Реализация сортировки пузырьком

def bubble(arr, n):  
 i = 0  
 while i < n - 1:  
 j = i + 1  
 while j < n:  
 if arr[i] > arr[j]:  
 tmp = arr[i]  
 arr[i] = arr[j]  
 arr[j] = tmp  
 j += 1  
 i += 1

Листинг 2. Реализация сортировки выбором

def selection(arr, n):  
 i = 0  
 while i < n - 1:  
 j = i + 1  
 mina = arr[i]  
 mini = i  
 while j < n:  
 if arr[j] < mina:  
 mina = arr[j]  
 mini = j  
 j += 1  
 tmp = arr[i]  
 arr[i] = arr[mini]  
 arr[mini] = tmp  
 i += 1  
 return arr

Листинг 3. Реализация сортировки вставкой

def insert(arr, n):  
 i = 0  
 while i < n:  
 j = i - 1  
 key = arr[i]  
 while arr[j] > key and j >= 0:  
 arr[j + 1] = arr[j]  
 j -= 1  
 arr[j + 1] = key  
 return arr

Листинг 4. Функция подсчета среднего времени выполнения алгоритма сортировки для случайного массива длиной size.

def time\_analyze(function, iterations=100, size=5):  
 t1 = process\_time()  
 for \_ in range(iterations):  
 start(size, function, False)  
 t2 = process\_time()  
 return (t2 - t1) / iterations

## Тестирование функций

Для модульного тестирования реализованных алгоритмах (см. листинги 1-3) была использована стандартная библиотека языка python – unittest. Все функции протестированы на различных входных данных.

Листинг 5. Функция сравнения массивов

def compare\_arr(arr1, arr2):  
 if len(arr1) != len(arr2):  
 return False  
 for i in range(len(arr1)):  
 if arr1[i] != arr2[i]:  
 return False  
 return True

Листинг 6. Функция тестирования сортировок

def full\_test(func):  
 err = 0  
 *# Проверка алгоритма на массивах длиной от 1 до 100* for i in range(1, 100):  
 if not compare\_arr(start(i, func, False), sorted\_arr(i, 0)):  
 err += 1  
  
 *# Проверка алгоритма на ста отсортированных массивах* for i in range(1, 100):  
 if not compare\_arr(func(sorted\_arr(i, 0), i), sorted\_arr(i, 0)):  
 err += 1  
  
 *# Проверка алгоритма на ста обратно-отсортированных массивах* for i in range(1, 100):  
 if not compare\_arr(func(sorted\_arr(i, 1), i), sorted\_arr(i, 0)):  
 err += 1  
  
 if err:  
 print(**'Зафиксированно '**, err, **'ошибок для функции:'**, func.\_\_name\_\_)  
 else:  
 print(**'Все тесты пройдены успешно для функции:'**, func.\_\_name\_\_)

Листинг 7. Функция, создающая отсортированный или обратно-отсортированный массив

def sorted\_arr(size, start=0):  
 arr = []  
 if start == 0:  
 for i in range(size):  
 arr.append(i)  
 else:  
 for i in range(size):  
 arr.append(size - i - 1)  
 return arr

Все тесты были пройдены успешно для всех реализованных алгоритмов сортировки.

# Экспериментальная часть

При запуске программы первое, что видит пользователь – это интерфейс.

## Интерфейс

На рис. 4 представлено главное меню программы. В зависимости от выбранного пункта меню запускается соответствующий алгоритм

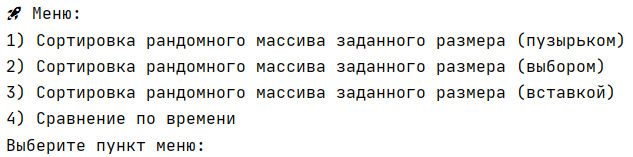


Рисунок 4. Интерфейс – меню

На рис. 4 - 8 приведены примеры работы программы при выборе пунктов меню 1-4.

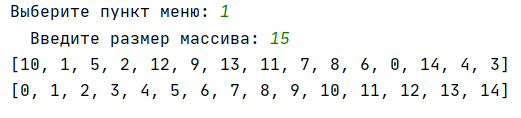


Рисунок 5. Пример работы программы при выборе пункта 1

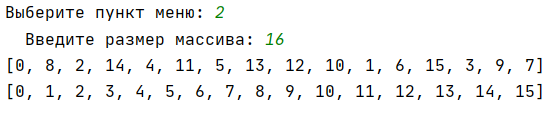


Рисунок 6. Пример работы программы при выборе пункта 2

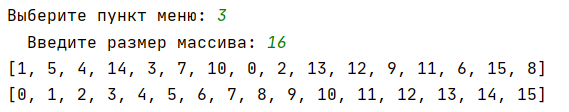


Рисунок 7. Пример работы программы при выборе пункта 3

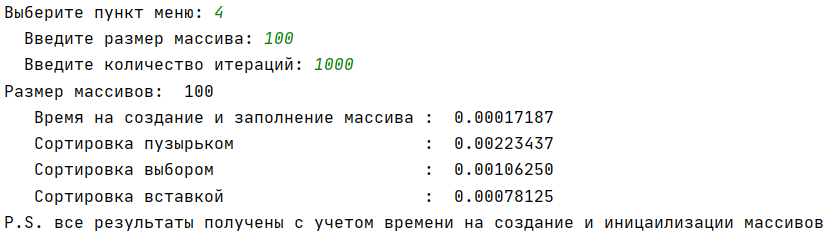


Рисунок 8. Пример работы программы при выборе пункта 4

## Сравнение алгоритмов по времени работы реализаций

Для сравнения в программе необходимо провести замеры процессорного времени для выполнения n – количества вычислений для массивов одинакового размера. Данное время замеров представлено в таблице 1.

Таблица 1. Сравнение алгоритмов по времени, произвольный случай

|  |  |  |
| --- | --- | --- |
| Сортировка | Размер массивов | Среднее время, сек |
| Пузырьком | 100 | 0.00217266 |
| Выбором | 100 | 0.00111016 |
| Вставкой | 100 | 0.00097969 |
| Пузырьком | 250 | 0.01693750 |
| Выбором | 250 | 0.00814063 |
| Вставкой | 250 | 0.00745312 |
| Пузырьком | 500 | 0.05468750 |
| Выбором | 500 | 0.02500000 |
| Вставкой | 500 | 0.02437500 |

Замеры времени усреднялись для каждого набора экспериментов. Для этого все вычисления производились на случайных массивах при 1000-10000 итераций. Количество итераций обратно пропорционально размеру массивов. Это связано с тем, что время сортировки больших массивов велико.

Далее посмотрим, как это время зависит от конкретных входных данных. Запустим наши алгоритмы для на произвольных, наилучших и наихудших входных данных. Результаты времени работы алгоритмов представлены в таблице 2.

Таблица 2. Сравнение времени на наилучших и наихудших входных данных

|  |  |  |
| --- | --- | --- |
| Сортировка | Размер массивов | Среднее время, сек |
| Пузырьком (произвольное) | 100 | 0.00167500 |
| Пузырьком (наилучшее) | 100 | 0.00140312 |
| Пузырьком (наихудшее) | 100 | 0.00251406 |
| Выбором (произвольное) | 100 | 0.00111016 |
| Выбором (наилучшее) | 100 | 0.00105469 |
| Выбором (наихудшее) | 100 | 0.00146719 |
| Вставкой (произвольное) | 100 | 0.00097969 |
| Вставкой (наилучшее) | 100 | 0.00003906 |
| Вставкой (наихудшее) | 100 | 0.00262031 |

**Вывод**: наглядно видно, что при работе с упорядоченными или практически упорядоченными следует использовать сортировку вставкой (при выборе из трех предложенных алгоритмов). Сортировка выбором показывает стабильный результат, практически независимый от входных данных, в отличии от сортировки вставкой, время которой сильно разбросано. Время сортировки пузырьком имеет меньший разброс, однако из-за того, что верхний предел совпадает с пределом сортировки вставкой, мы можем сказать, что сортировка вставкой в среднем будет лучше сортировки пузырьком.

Заключение

В ходе работы были изучены некоторые алгоритмы сортировки массивов. Реализованы 3 алгоритма: сортировка пузырьком, сортировка выбором и сортировка вставкой. Был приведен программный код реализации этих алгоритмов.

Была подсчитана трудоемкость каждого из алгоритмов. А также было проведено сравнение алгоритмов по времени и трудоемкости. Была определена зависимость времени работы алгоритма от входных данных.

Цель работы достигнута. Получены практические навыки реализации алгоритмов сортировки, а также проведена исследовательская работа по вычислению и сравнению трудоемкости на примере трех алгоритмов сортировки массивов.

Список литературы

1. Дж. Макконнелл. Анализ алгоритмов. Активный обучающий подход. –М.: Техносфера, 2017. – 267 c.

2. Официальный сайт Python, документация [Электронный ресурс]. Режим доступа: <https://docs.python.org/3/library/time.html>, свободный (дата обращения: 16.09.20).